

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**LEAST SQUARES SOLUTIONS IN STATISTICAL ORBIT
DETERMINATION USING SINGULAR VALUE
DECOMPOSITION**

by

Patrick M. Marshall

June 1999

Thesis Advisor:
Thesis Co-Advisor:

Donald A. Danielson
David Canright

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 4

19990915 067

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 1999		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE LEAST SQUARES SOLUTIONS IN STATISTICAL ORBIT DETERMINATION USING SINGULAR VALUE DECOMPOSITION				5. FUNDING NUMBERS
6. AUTHOR(S) Marshall, Patrick M.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000				8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSORING / MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.				12b. DISTRIBUTION CODE
13. ABSTRACT (maximum 200 words) This thesis is a partial analysis of the Naval Space Command statistical orbit determination algorithms. Through a process called Differential Correction, data from space surveillance radar observation stations is synthesized with previously accumulated element sets to maintain accurate orbital object position information. Differential Correction is a nonlinear least squares process employing statistical techniques to minimize the residual measurement error thereby increasing relative position information accuracy. This study focuses specifically on the algorithmic methods of solution to the systems of normal equations generated by the Differential Correction process. A comparison and analysis of the present Naval Space Command method and the singular value decomposition method is presented. Algorithmic constructions are presented for both methods and problematic areas are highlighted. The principal focus herein is to demonstrate the benefit of singular value decomposition when attempting to solve systems of equations whose coefficient matrices are dense and nearly singular. These results generalize to commonly employed normal equation solution algorithms and are intended for further study and possible incorporation by Naval Space Command as part of future modernization plans.				
14. SUBJECT TERMS Least Squares, Nonlinear Least Squares, Normal Equations, Singular Value Decomposition, Gaussian Elimination, Differential Correction				15. NUMBER OF PAGES 62
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Approved for public release; distribution is unlimited

**LEAST SQUARES SOLUTIONS IN STATISTICAL ORBIT
DETERMINATION USING SINGULAR VALUE DECOMPOSITION**

Patrick M. Marshall
Captain, United States Army
B.S., Augusta State University, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

**NAVAL POSTGRADUATE SCHOOL
June 1999**

Author: _____


Patrick M. Marshall

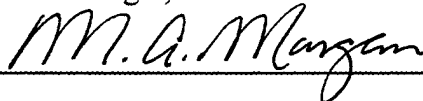
Approved by: _____



Donald A. Danielson, Thesis Advisor



David Canright, Thesis Co-Advisor



Michael Morgan, Chairman
Department of Mathematics

ABSTRACT

This thesis is a partial analysis of the Naval Space Command statistical orbit determination algorithms. Through a process called Differential Correction, data from space surveillance radar observation stations is synthesized with previously accumulated element sets to maintain accurate orbital object position information. Differential Correction is a nonlinear least squares process employing statistical techniques to minimize the residual measurement error thereby increasing relative position information accuracy. This study focuses specifically on the algorithmic methods of solution to the systems of normal equations generated by the Differential Correction process. A comparison and analysis of the present Naval Space Command method and the singular value decomposition method is presented. Algorithmic constructions are presented for both methods and problematic areas are highlighted. The principal focus herein is to demonstrate the benefit of singular value decomposition when attempting to solve systems of equations whose coefficient matrices are dense and nearly singular. These results generalize to commonly employed normal equation solution algorithms and are intended for further study and possible incorporation by Naval Space Command as part of future modernization plans

TABLE OF CONTENTS

I. INTRODUCTION	1
II. BACKGROUND	3
A. STATISTICAL ORBIT DETERMINATION	3
B. DERIVATION OF THE NORMAL EQUATIONS	3
C. GENERAL LINEAR LEAST SQUARES	3
D. LEAST SQUARES APPLIED TO ORBITAL MECHANICS	6
III. NORMAL EQUATION SOLUTION METHODS	11
A. GENERAL	11
B. NAVSPACECOM GAUSS-JORDAN ELIMINATION	12
C. SINGULAR VALUE DECOMPOSITION	16
IV. COMPARISON OF METHODS	27
A. GENERAL	27
B. SENSITIVITY ANALYSIS	28
C. ERROR ANALYSIS	32
D. COMPUTATIONAL EFFICIENCY	35
V. CONCLUSIONS	37
A. SUMMARY OF FINDINGS	37
B. RECOMMENDATION	38
C. CONCLUSION	38
APPENDIX A: TEST MATRIX TYPE 1	41
APPENDIX B: TEST MATRIX TYPE 2	45
LIST OF REFERENCES	49
INITIAL DISTRIBUTION LIST	51

ACKNOWLEDGEMENTS

Foremost I thank my wife Pamela for her constant encouragement and endless support of my professional life, my children for their continuous patience with the demands of my career, and my mother whose life-long support has inspired me thus far.

A special gratitude I give to Professor Don Danielson for his complete inspiration through endless hours of scientific education and philosophical contemplation whose constant encouragement and complete freedom empowered my pursuit, my education.

I also thank Professor David Canright for sharing his exceptional eye for detail, providing absolutely clairvoyant explanation of the most tedious algorithm, and most of all for his friendship and steadfastness in helping me complete the task at hand.

Additionally, I would like to thank Professors Bill Gragg, Dick Franke, Carlos Borges, Chris Frenzen, and Maury Weir for always selflessly lending their extra time.

And finally to Jhoie Pasadilla, always there in all things small never failing to smile through it all - Thank you.

I. INTRODUCTION

The Naval Space Command maintains a database of element sets for more than 9,000 objects. Through a process called *Differential Correction*, data from space surveillance observation stations is synthesized with previously accumulated element sets. Differential Correction is a nonlinear least squares process employing statistical techniques which provide for accurate orbit state estimation. Radar measurements of an object's motion are collected by dispersed observation stations and passed to NAVSPACECOM for central processing. NAVSPACECOM receives daily approximately 270,000 new observation sets and uses them to update as many as 18,000 element sets. NAVSPACECOM performance reports indicate that approximately 98.5% of the element sets get updated without manual intervention by their computer software called AUTODC. The remaining 1.5% must be manually analyzed. Differential correction is a highly complex step-wise process that entails more than 16 separate mathematical computations. Even with automated support, this constitutes a significant work load.

Numerical linear algebra is at the core of the differential correction process. In general, data is accumulated, normal equations are formed, and numerical linear algebra routines are called to solve the matrix system of normal equations. The method of normal equation solution and its associated algorithms are the focus of this analysis. The purpose of this thesis is to demonstrate the benefits of using singular value decomposition when obtaining least squares solution to systems of normal equations.

II. BACKGROUND

A. STATISTICAL ORBIT DETERMINATION

The NAVSPACECOM differential correction method is a sequential batch nonlinear least squares statistical process. This is an iterative method which refines the stored orbital element sets by applying state adjustments obtained from differential correction of the current orbital observations. These state updates are tested for fit in the least squares sense and if acceptable applied to the previous nominal state then stored as the new nominal orbital element set. The fundamental mathematical steps are now outlined; for further details, see Vallado [1] and Danielson and Canright [2].

B. DERIVATION OF THE NORMAL EQUATIONS

These descriptions have been simplified and are only intended to provide sufficient background for the purpose of detailing the algorithmic composition and term-wise structure of the normal equation system. For a very detailed description of the entire set of NAVSPACECOM procedures refer to Danielson and Canright [2].

C. GENERAL LINEAR LEAST SQUARES

We begin the process by noting that our goal is to solve the generally inconsistent system of equations whose matrix representation is

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

by minimizing the residual, \mathbf{r} , where

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x} \quad (2)$$

The method of least squares can be applied to solve such linear systems. Typically these systems are highly overdetermined. They have many more rows (m), or equations than columns (n) or unknowns and as such are inconsistent. From elementary linear algebra [3], we know that the *range* of the matrix $R(\mathbf{A})$ is the orthogonal complement of the *null space* of its transpose $N(\mathbf{A}^T)$. As such, the multiplication \mathbf{A}^T always produces a consistent set of n equations and n unknowns. The only issue with this method however arises when some set of the variables from the original linear system are correlated. If this happens, the newly formed consistent system will fail to have a unique solution.

The general method proceeds as follows. Left multiplication of equation (1) yields,

$$\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b} \quad (3)$$

Equation (2) implies

$$\mathbf{A}^T \mathbf{r} = \mathbf{A}^T (\mathbf{b} - \mathbf{A} \mathbf{x}) = \mathbf{0} \quad (4)$$

Equations from the original system may be weighted by applying a diagonal weighting matrix. This expands as

$$\mathbf{A}^T \mathbf{W} \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{W} \mathbf{b} \quad (5)$$

Residuals from the new, weighted system satisfy

$$\mathbf{A}^T \mathbf{W} \mathbf{r} = \mathbf{A}^T \mathbf{W} (\mathbf{b} - \mathbf{A} \mathbf{x}) = 0 \quad (6)$$

Provided $\mathbf{A}^T \mathbf{W} \mathbf{A}$ is nonsingular, we obtain the solution as,

$$\mathbf{x} = (\mathbf{A}^T \mathbf{W} \mathbf{A})^{-1} \mathbf{A}^T \mathbf{W} \mathbf{b} \quad (7)$$

This result commonly referred to as the *normal equations* forms the basis of the least squares solution process.

Differential Correction employs *sequential batch* techniques when acquiring the current state solution from different observational data input sets. This method synthesizes the separate batch solutions as follows.

Given the systems for two batches as

$$\mathbf{A}_1 \mathbf{x} = \mathbf{b}_1 \text{ and } \mathbf{A}_2 \mathbf{x} = \mathbf{b}_2$$

In order for the single solution to be meaningful, these batch systems must be simultaneously solved. This is accomplished by forming the weighted least squares system as follows.

$$[(\mathbf{A}_1^T \mathbf{W} \mathbf{A}_1) + (\mathbf{A}_2^T \mathbf{W} \mathbf{A}_2)] \mathbf{x} = (\mathbf{A}_1^T \mathbf{W} \mathbf{b}_1) + (\mathbf{A}_2^T \mathbf{W} \mathbf{b}_2)$$

The process can be simplified slightly by reusing the solution from \mathbf{x}_1

$$[(\mathbf{A}_1^T \mathbf{W} \mathbf{A}_1) + (\mathbf{A}_2^T \mathbf{W} \mathbf{A}_2)] \mathbf{x} = (\mathbf{A}_1^T \mathbf{W} \mathbf{A}_1) \mathbf{x}_1 + (\mathbf{A}_2^T \mathbf{W} \mathbf{b}_2)$$

When acquiring the weighted solution simultaneously, different weights may be applied to the different sets of equations either by reducing \mathbf{W}_1 or by scaling $(A^T \mathbf{W} A)_1$. Notice here that the batching process does not significantly alter the basic least squares problem nor does it alter the complication arising when some portion of the left hand side constitutes a singular matrix.

D. LEAST SQUARES APPLIED TO OBITAL MECHANICS

The reader is directed to Danielson and Canright [2] for extensive mathematical description and existing code documentation from which the following outline extract was taken.

- (i) Assume an initial nominal state

$$\mathbf{X}_{\text{nominal}} = \begin{bmatrix} X_1 \\ \vdots \\ X_8 \end{bmatrix}.$$

- (ii) Compute the values of the observed parameters Y_c at N times corresponding to the observations Y_o (each observation set contains at a maximum 6 numbers)

$$\mathbf{Y}_o = \begin{bmatrix} Y_{o_1} \\ \vdots \\ Y_{o_6} \end{bmatrix}_i, \quad \mathbf{Y}_c = \begin{bmatrix} Y_{c_1} \\ \vdots \\ Y_{c_6} \end{bmatrix}_i, \quad \text{where } i = 1, \dots, N$$

- (iii) Compute the residuals or "O-Cs" $(Y_o - Y_c)_i$ (observed minus calculated parameters). Arrange these as the $6N \times 1$ column matrix

$$\mathbf{b} = \begin{bmatrix} (Y_{o_1} - Y_{c_1})_i \\ (Y_{o_2} - Y_{c_2})_i \\ \vdots \\ (Y_{o_{61}} - Y_{c_6})_i \end{bmatrix}$$

(iv) Calculate the partial derivatives

$$\frac{\partial Y_c}{\partial X} = \frac{\partial Y_c}{\partial(\mathbf{r}, \mathbf{v})} \frac{\partial(\mathbf{r}, \mathbf{v})}{\partial X}$$

$$= \begin{bmatrix} \frac{\partial Y_{c_1}}{\partial r_I} & \frac{\partial Y_{c_1}}{\partial r_J} & \frac{\partial Y_{c_1}}{\partial r_K} & \frac{\partial Y_{c_1}}{\partial v_I} & \frac{\partial Y_{c_1}}{\partial v_J} & \frac{\partial Y_{c_1}}{\partial v_K} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial Y_{c_6}}{\partial r_I} & \frac{\partial Y_{c_6}}{\partial r_J} & \frac{\partial Y_{c_6}}{\partial r_K} & \frac{\partial Y_{c_6}}{\partial v_I} & \frac{\partial Y_{c_6}}{\partial v_J} & \frac{\partial Y_{c_6}}{\partial v_K} \end{bmatrix} \begin{bmatrix} \frac{\partial r_I}{\partial X_1} \dots \frac{\partial r_I}{\partial X_8} \\ \frac{\partial r_J}{\partial X_1} \dots \frac{\partial r_J}{\partial X_8} \\ \frac{\partial r_K}{\partial X_1} \dots \frac{\partial r_K}{\partial X_8} \\ \frac{\partial v_I}{\partial X_1} \dots \frac{\partial v_I}{\partial X_8} \\ \frac{\partial v_J}{\partial X_1} \dots \frac{\partial v_J}{\partial X_8} \\ \frac{\partial v_K}{\partial X_1} \dots \frac{\partial v_K}{\partial X_8} \end{bmatrix}$$

at each observation time and arrange the coordinates of the position and velocity vectors,

(r_I, r_J, r_K) and (v_I, v_J, v_K) , into the following $6N \times 8$ matrix.

$$\mathbf{A} = \begin{bmatrix} \left(\frac{\partial Y_{c_1}}{\partial X_1} \right)_1 & \dots & \left(\frac{\partial Y_{c_1}}{\partial X_8} \right)_1 \\ \left(\frac{\partial Y_{c_2}}{\partial X_1} \right)_1 & \dots & \left(\frac{\partial Y_{c_2}}{\partial X_8} \right)_1 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

(v) Form the normal equations:

$$\mathbf{A}^T \mathbf{W} \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{W} \mathbf{b}$$

Where \mathbf{W} denotes a $6N \times 6N$ weighting matrix.

Finally

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_8 \end{bmatrix}$$

are the differential corrections. Note that $\mathbf{A}^T \mathbf{W} \mathbf{A}$ is an 8×8 matrix, and that $\mathbf{A}^T \mathbf{W} \mathbf{b}$ is an 8×1 matrix.

(vi) Solve the normal equations:

$$\mathbf{x} = (\mathbf{A}^T \mathbf{W} \mathbf{A})^{-1} \mathbf{A}^T \mathbf{W} \mathbf{b}$$

(vii) Update the elements:

$$\mathbf{X}_{new} = \mathbf{X}_{old} + \mathbf{x}$$

(For the first iteration, \mathbf{X}_{old} is $\mathbf{X}_{nominal}$.)

(viii) Lastly, apply the following RMS test to determine if iterations should continue.

$$\left| \frac{RMS_i - RMS_{i+1}}{RMS_i} \right| \leq \varepsilon$$

where,

$$RMS_i = \sqrt{\frac{1}{N-1} \sum_{i=1}^N r_i^2} = \sqrt{\frac{\mathbf{b}^T \mathbf{b}}{N-1}}$$

Generally, in nonlinear least squares our goal is to minimize the sum of the residuals squared. It is therefore appropriate to use some measure of this as stopping

criteria. In the differential correction process, simply cease iteration when the RMS "stops" changing. Since all of the necessary input to the RMS equations has been accumulated this proves an efficient method as well.

III. NORMAL EQUATION SOLUTION METHODS

A. GENERAL

At present the Naval Space Command, Differential Correction process relies on Gauss-Jordan elimination with full pivoting for solution to the normal equations. The subroutines responsible for processing the complete normal equation solutions are named AMA06, AMA04 and AMA03. After the appropriate Differential Correction algorithms have formed the standard system of normal equations as described in the previous section, AMA06 is called to read in the initial array entries, form the matrix $E = A^T W A$, duplicate it and begin preconditioning processes designed to identify singular matrices. Following sufficient preconditioning, AMA06 then calls AMA04 which in turn computes E^{-1} . Finally, AMA03 is called to apply Gauss-Jordan elimination and solve, $x = (A^T W A)^{-1} A^T W b$. The 8x8 matrix $A^T W A$ is input as E and the 8x1 matrix $A^T W b$ is input as G.

The following section describes specific processes and algorithms used to solve the normal equations with Gaussian elimination and with Singular Value Decomposition, SVD. Each process is initiated with the previous normal equation derivation. The present NAVSPACECOM differential correction process uses the Gauss Jordan elimination method with full pivoting.

Focusing back on step (vi) of the least squares orbital mechanics process, *solution of the normal equations*, we now compare the present NAVSPACECOM method with Singular Value Decomposition method. The refined problem is now, *how to acquire the best approximation at each step of the nonlinear approximation process so as to ensure maximum efficiency of the iterative routine while minimizing the associated rounding error in the next iterate?*

B. NAVSPACECOM GAUSS-JORDAN ELIMINATION

The Naval Space Command uses Gauss-Jordan elimination with full pivoting for solution to the system of normal equations. Their algorithm is very similar to the one in the book, Numerical Recipes, by Press, et al. [7].

Typical Gauss-Jordan with Full Pivoting

Every Gauss-Jordan step is a left multiplication by an elementary matrix, [3]. An overview of the general method of Gauss-Jordan elimination is as follows:

- (i) Augment the right hand side with the $n \times n$ identity matrix.
- (ii) Perform elementary operations on the augmented, $n \times 2n$ system until the $n \times n$ identity matrix is reformed in the first n columns of the matrix
- (iii) Recall the last n columns of the augmented matrix as the inverse.

This method is usually done *in place*; that is, the actual augmentation is never performed. The algorithm simply replaces the original input matrix, *in place*, with its inverse. For this reason, if there is ever a need to recall the original data, a copy of the original matrix must be made on input.

Full pivoting is the process by which rows and columns of the original matrix are reordered so that the element largest in magnitude is moved to the upper left corner of the matrix. Then, each successive pivot row is maneuvered similarly. Why do this? In the computational process, the subsequent rows below the pivot in question are being reduced to zero by dividing through by a scaled multiple of that pivot. If the process is not begun in this manner, that is to say if the element below the pivot is greater in magnitude than the pivot, the pivot must be scaled by increasing its magnitude. The scaling factor is the element being zeroed out divided by the pivot, and if the pivot is small enough this approaches division by zero. In finite precision arithmetic, this can lead to a divide by zero condition. If the difference in magnitude of the pivot is sufficiently less than the element being *zeroed out*, the computer can evaluate the expression and return "Not-A-Number". Even if the division by zero extreme condition does not occur, scaling the matrix through by these sufficiently small quantities inflates the rounding error. How does pivoting protect against this? By always scaling so that each successive pivot is the next largest in magnitude, the previous process of dividing the subdiagonal elements, results in scaling the pivot row by decreasing its magnitude. This has the effect of moving the division operations away from the $1/\text{zero}$ condition and deflates the magnitude of rounding error to the minimum possible given the actual element-wise composition of the matrix.

Partial pivoting is a similar process that allows only row exchanges. While this may appear to be a sound concept with less computational overhead, in practice, matrix elements, not directly below (in some other column) the pivot being operated on, may still be sufficiently greater in value than subsequently available pivot choices. If this occurs, the solution is again driven toward a divide by zero condition. By manipulating both rows and columns, as in full pivoting, we reduce the likelihood of the divide by zero condition to the greatest extent possible.

When employing the full pivoting routines, the general Gauss-Jordan process is augmented by left and right permutation matrices which track the pivot maneuvers. The corresponding information is stored as an array and applied during the back solve process thereby ensuring correct association of the coefficients and variables from the original system of equations.

It must be noted that the previous pivoting processes only reduce the likelihood of algorithm failure. They do not prevent it. In the event of near singular, noninvertible conditions, the Gauss-Jordan full pivoting routine has no mechanism to correct or compensate. This is an inherent critical deficiency. Failure due to nearly singular input matrices routinely occurs when processing least squares systems that are derived from closely sampled data.

Unique features of NAVSPACECOM Gauss-Jordan

The following system processes are extracted as reported in the code documentation research of Danielson and Canright, [2]. The matrix representation of the system of normal equations is loaded as the input matrix E . The input matrix is symmetric positive semi-definite by its inherent construction. As such, only values on and to the right of the diagonal are read as input. The below diagonal elements are copied from their respective symmetric counterparts. At this point a preconditioning routine, AMA06, is called. It samples the matrix to determine if any off-diagonal element of E is too large relative to its corresponding diagonal elements. AMA06 tests each $E_{ij}^2 > (\text{SING})E_{ii}E_{jj}$, where SING is a parameter value. If any E_{ij}^2 violates this inequality then that row (for even-number calls) or column (for odd-number calls) is "*inactivated*". The threshold SING is initially set to near one. Then, if the active matrix is singular, the threshold SING is lowered by 10% (attempting to inactivate more rows/columns) and the solution is tried again. At this point, a saved copy of the original matrix must be reloaded. The immediate effect of this process is to eliminate divide by zero conditions when pivot elements are very small. Eventually, either a solution is found, or the whole matrix is made inactive (flagged by a return value $\text{SING} = 0$), or the threshold gets too small ($\text{SING} \leq 0.01$) indicating that the entire system is singular. In the event that no matrix remains, an error condition is returned and all further processing of this set of observational data ceases. A tally of these failures is presented as an output report in the batch run statistics.

There is a fundamental problem with this preconditioning methodology. Under no circumstances are the root causes of the perceived singularities dealt with. Further, in terms of computational effort, the computer is permitted to cycle excessively without sufficient intermediate checks to determine if the initial perceived singular conditions are removable.

In summary, Gaussian elimination is generally the most computationally efficient numerical method available; however, it suffers from numerical instability under certain conditions, which are detailed in Chapter IV, Comparison of Methods. Unfortunately, these very same conditions generally arise in systems of normal equations. As noted earlier, whenever correlation of variables from the original system equations occurs, the resulting matrix $\mathbf{A}^T \mathbf{A}$ represents a consistent set of equations with redundant solutions. These redundant solutions drive the matrix toward a singular condition.

C. SINGULAR VALUE DECOMPOSITION

The Singular Value Decomposition, SVD of an arbitrary $m \times n$ matrix is the factorization of \mathbf{A} into $\mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$, where \mathbf{U} and \mathbf{V} are orthogonal matrices and $\mathbf{\Sigma}$ is the rectangular $m \times n$ matrix whose first r rows form a square, diagonal submatrix with elements $\sigma_1 \cdots \sigma_r$, i.e., the singular elements of \mathbf{A} with the remainder of $\mathbf{\Sigma}$ being zeros.

The process proceeds generally as follows (See [6]):

- (i) Reduce the general input matrix \mathbf{A} to bidiagonal form \mathbf{B} with orthogonal matrices \mathbf{U}_1 and \mathbf{V}_1 where $\mathbf{A} = \mathbf{U}_1 \mathbf{B} \mathbf{V}_1^T$. \mathbf{B} is nonzero only on its main

diagonal and first super diagonal. This is accomplished with Householder transformations in the actual algorithms.

- (ii) Find the SVD of \mathbf{B} , $\mathbf{B} = \mathbf{U}_2 \Sigma \mathbf{V}_2^T$, where Σ is the diagonal matrix of singular values and \mathbf{U}_2 and \mathbf{V}_2 are orthogonal matrices whose columns are the respective left and right singular vectors. The Gram-Schmidt process produces this result.
- (iii) Combine these decompositions to form $\mathbf{A} = (\mathbf{U}_1 \mathbf{U}_2) \Sigma (\mathbf{V}_1 \mathbf{V}_2)^T$. The columns of $\mathbf{U} = (\mathbf{U}_1 \mathbf{U}_2)$ and $\mathbf{V} = (\mathbf{V}_1 \mathbf{V}_2)$ are the respective left and right singular vectors of \mathbf{A} .

Step (i) reduces the \mathbf{A} matrix to bidiagonal form by applying Householder transformations on both left and right sides. The symmetry of the original matrix is preserved throughout. Step (ii) employs the Gram-Schmidt process to orthogonalize the columns of \mathbf{U}_2 and \mathbf{V}_2 . Step (iii) reforms the matrix factors. Specific algorithms for these subordinate functions are detailed in Golub [5] and Demmel [6]. The SVD algorithm simply incorporates them. Another extremely useful secondary benefit of the SVD is provided by its fundamental structure. That is the factor matrices \mathbf{U} and \mathbf{V} have a very special structure. They are *orthogonal*. Where, $\mathbf{A} = \mathbf{U} \Sigma \mathbf{V}^T$, the columns of \mathbf{U} are the eigenvectors of $\mathbf{A} \mathbf{A}^T$ and the columns of \mathbf{V} are the eigenvectors of $\mathbf{A}^T \mathbf{A}$, Strang, [3]. Further, orthogonal matrices have the nicety that $\mathbf{U}^T \mathbf{U} = \mathbf{U} \mathbf{U}^T = \mathbf{I}$. This infers

directly that when solving the system of normal equations,

$\mathbf{Ax} = \mathbf{b}$ as $\mathbf{U} \Sigma \mathbf{V}^T \mathbf{x} = \mathbf{b}$ the respective transposes, \mathbf{U}^T and \mathbf{V} , are left multiplied yielding $\mathbf{x} = \mathbf{V} \Sigma^+ \mathbf{U}^T \mathbf{b}$. Interestingly enough, no actual inverse matrices have to be calculated. Further, the transpose of \mathbf{U} and \mathbf{V} can be done in place. The real benefit is the complete absence of rounding error when taking the transpose of a matrix.

The problem with the Gauss-Jordan method is the requirement for the columns of \mathbf{A} to be independent. As described in detail in the previous section, when ill-conditioned matrices are input, a great deal of computational effort is required in preconditioning the matrix. Without this preconditioning, even nonsingular but very ill-conditioned matrices may cause algorithms to break down. The computer simply perceives the matrix to be singular to its level of machine precision.

The key to SVD's computational stability lies in its orthogonality. Matrix rank problems arise frequently in computer arithmetic. Determining the rank of an ill-conditioned matrix can be challenging in the presence of roundoff error and noisy data. The SVD allows for practical dealing with numerical rank deficiency

The following theorem is taken from Golub, [5]. See [5] for a detailed proof. The theorem provides the detail necessary for determining how "close" the given \mathbf{A} matrix is to one of lower rank. The 2-norm here is the matrix derivation of the vector Euclidean norm [3] defined as follows.

$$\text{least upper bound}(\mathbf{A}) = \max \left(\frac{\|\mathbf{Ax}\|}{\|\mathbf{x}\|} \right) \quad \text{where} \quad \|\mathbf{A}\| \leq \text{least upper bound}(\mathbf{A}) \|\mathbf{x}\|$$

Theorem

Let the SVD of $\mathbf{A} \in \mathbf{R}^{m \times n}$ be given. If $k < r = \text{rank}(\mathbf{A})$, and

$$\mathbf{A}_k = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

then,

$$\min_{\text{rank}(\mathbf{B}) = k} \|\mathbf{A} - \mathbf{B}\|_2 = \|\mathbf{A} - \mathbf{A}_k\|_2 = \sigma_{k+1}$$

This striking result offers a method of computation using stored values which specifically reveal the magnitude of the degenerate numerically singular condition of the \mathbf{A} matrix. The details of this theorem indicate that the *smallest singular value* of \mathbf{A} is the 2-norm distance of \mathbf{A} to the set of all rank deficient matrices. Iteration is no longer required to determine the degree of singularity.

Now, let's look at an example of how to make the most out of ill-conditioned least squares systems with the SVD. See Strang [3] for more on this. In general, the least squares problem has one very stringent requirement, the columns of the \mathbf{A} matrix must be independent or the rank of \mathbf{A} must be equal to n , the number of columns. This is often referred to as *full rank*. If not, \mathbf{A} is not invertible then $\mathbf{Ax} = \mathbf{b}$ can not determine \mathbf{x} . As described earlier, any vector from the null space of \mathbf{A} can be added to \mathbf{x} . Now let's examine what happens. There are two possible situations, either the rows of \mathbf{A} may be dependent or the columns of \mathbf{A} may be dependent. The first situation implies the system of equations may have no solution and the second situation implies that any solution is not unique. The dependent column case makes this a particularly difficult yet interesting

problem. As discussed earlier, when we have dependent rows the solution we seek may be outside the column space of \mathbf{A} . Our course of action now becomes simply project \mathbf{b} onto the column space of \mathbf{A} . Now the greater challenge. After making that projection we find \mathbf{A} has dependent columns and the solution is not unique. At this point, we must now employ the criteria for selecting the optimal solution and choose the one with *minimum length*.

Consider the following example:

$$\mathbf{A} = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

where \mathbf{A} is diagonal with dependent rows and columns.

Here we see the columns all end in 0. As per case (i), the closest vector to $\mathbf{b} = (b_1, b_2, b_3, b_4)$ is $\mathbf{p} = (b_1, b_2, 0, 0)$ the projection onto the column space of \mathbf{A} . The magnitude of error here is $\mathbf{b} = (0, 0, b_3, b_4)$ the perpendicular to the columns. The best solution now is attained when we solve the first two equations. Since the last two equations indicate $0 = b_3$ and $0 = b_4$, the error in those equations cannot be reduced but the error in the first two will be zero.

$$\mathbf{A}\bar{\mathbf{x}} = \mathbf{p} \text{ is } \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ 0 \\ 0 \end{bmatrix} \quad (8)$$

Now the challenge, the dependent columns imply $\bar{\mathbf{x}}$ is not unique! The first two components are $\frac{\sigma_1}{b_1}$ and $\frac{\sigma_2}{b_2}$, but x_3 and x_4 are completely arbitrary. Now apply the minimum length criteria and see that these arbitrary components must be identically zero to attain the best approximation.

That is,

$$\mathbf{x}^+ = \begin{bmatrix} \frac{b_1}{\sigma_1} \\ \frac{b_2}{\sigma_2} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \quad (9)$$

The *minimum length solution* to $\mathbf{A}\bar{\mathbf{x}} = \mathbf{p}$ is, \mathbf{x}^+ , Strang [3]. Again, the useful result is specifically the equation that reveals \mathbf{x}^+ . This process displays the matrix, which yields the desired result,

$$\text{If } \mathbf{A} = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ then } \mathbf{A}^+ = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } \mathbf{x}^+ = \mathbf{A}^+ \mathbf{b} = \begin{bmatrix} \frac{b_1}{\sigma_1} \\ \frac{b_2}{\sigma_2} \\ 0 \\ 0 \end{bmatrix} \quad (10)$$

\mathbf{A}^+ is referred to as the *pseudoinverse* and is the matrix which provides for solution to the nearly singular system of $\mathbf{Ax} = \mathbf{b}$, Strang, [3].

This entire process has one sticking point. Σ^+ is the pseudoinverse described earlier. Now, the magnitude of rounding error in applying the inverse process is limited to the sum of the errors when inverting the individual diagonal elements, σ_i . Each σ_i is the square root of each nonzero eigenvalue λ_i from both \mathbf{AA}^T and $\mathbf{A}^T\mathbf{A}$. Now, the fine point, what happens when σ_i is sufficiently small to induce a divide by zero condition when taking the pseudoinverse? The reciprocal of σ_i is set to zero by the code. Press, et. al., [7], denote this procedure as *editing* the singular values. The logic is sound. Recall the original formulation of the linear system. When redundant solutions (singular conditions) are encountered as a result of variable correlation, the matrix is unable to distinguish between the different basis functions and the associated distribution of the input data. By setting the reciprocal of any sufficiently close to zero singular value to zero, we effectively add a zero multiple to the fitting parameters as opposed to some large combination of the basis functions that are degenerate to the best fit, [7]. Further, if any nonzero singular value is very small, its reciprocal should also be set to zero. This term is most likely residual from rounding error and detracts from the optimal solution.

A rule for determining the editing tolerance of singular values is given by Press, et. al., [7] as follows. Set to zero any singular value, σ_i when,

$$\text{The ratio } \frac{\sigma_i}{\sigma_{\max}} < N\varepsilon$$

where N is the column length dimension and ε is the machine precision.

Consider the following example least squares problem which illustrates the mathematical principles of the algorithm. In this example \mathbf{A} is assumed symmetric on input just as in the Differential Correction form.

$$\text{Let } \mathbf{A} = \begin{bmatrix} 3 & -2 & 2 \\ -2 & 4 & 0 \\ 2 & 0 & 2 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

where

$$\mathbf{A}^T \mathbf{A} = \begin{bmatrix} 17 & -14 & 10 \\ -14 & 20 & -4 \\ 10 & -4 & 8 \end{bmatrix}$$

The eigenvalues

$$\begin{aligned} \lambda_1 &= 36 = 6^2 \Rightarrow \sigma_1 = 6 \\ \mathbf{A}^T \mathbf{A} - \lambda \mathbf{I} \text{ are } \lambda_2 &= 9 = 3^2 \Rightarrow \sigma_2 = 3 \\ \lambda_3 &= 0 = 0^2 \Rightarrow \sigma_3 = 0 \end{aligned}$$

whose corresponding eigenvectors are

$$\mathbf{v}_1 = \frac{1}{3} \begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix}, \mathbf{v}_2 = -\frac{1}{3} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}, \mathbf{v}_3 = \frac{1}{3} \begin{bmatrix} 2 \\ 1 \\ -2 \end{bmatrix}$$

Now $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ or $\mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{\Sigma}$ so

$$\mathbf{u}_1 = \frac{1}{\sigma_1} \mathbf{A} \mathbf{v}_1, \quad \mathbf{u}_2 = \frac{1}{\sigma_2} \mathbf{A} \mathbf{v}_2$$

$$\mathbf{u}_1 = \frac{1}{6} \begin{bmatrix} 3 & -2 & 2 \\ -2 & 4 & 0 \\ 2 & 0 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix} \frac{1}{3} = \frac{1}{18} \begin{bmatrix} 12 \\ 12 \\ 6 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix}$$

Similarly

$$\mathbf{u}_2 = -\frac{1}{3} \begin{bmatrix} 3 & -2 & 2 \\ -2 & 4 & 0 \\ 2 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} - \frac{1}{3} = \frac{1}{9} \begin{bmatrix} 3 \\ 3 \\ 6 \end{bmatrix} = -\frac{1}{3} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

Now for the zero singular value we must solve the homogeneous problem

$$\mathbf{A}^T \mathbf{u}_3 = 0 \text{ for } \mathbf{u}_3.$$

$$\left[\begin{array}{ccc|c} 3 & -2 & 2 & 0 \\ -2 & 4 & 0 & 0 \\ 2 & 0 & 2 & 0 \end{array} \right] \rightarrow \left[\begin{array}{ccc|c} 1 & -\frac{2}{3} & \frac{2}{3} & 0 \\ 0 & 1 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \end{array} \right] \Rightarrow u_3 \text{ is arbitrary}$$

$$\text{but, } u_2 = \frac{1}{2} u_3 \quad \text{and} \quad u_1 = -u_3$$

$$\text{so, } \mathbf{u}_3 = u_3 \begin{bmatrix} -1 \\ -\frac{1}{2} \\ 1 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 2 \\ 1 \\ -2 \end{bmatrix}$$

Now lets refit the components.

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix} \begin{bmatrix} 6 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \end{bmatrix} = \begin{bmatrix} 6\mathbf{u}_1 & 3\mathbf{u}_2 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \end{bmatrix} = \begin{bmatrix} 6\mathbf{u}_1 \mathbf{v}_1^T & 3\mathbf{u}_2 \mathbf{v}_2^T & 0\mathbf{v}_3^T \end{bmatrix}$$

This is simply the sum of rank one matrices.

Now,

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \Rightarrow \begin{bmatrix} \frac{2}{3} & \frac{-1}{3} \\ \frac{-2}{3} & \frac{-2}{3} \\ \frac{1}{3} & \frac{-2}{3} \end{bmatrix} \begin{bmatrix} 6 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} \frac{2}{3} & \frac{-2}{3} & \frac{1}{3} \\ \frac{-1}{3} & \frac{-2}{3} & \frac{-2}{3} \end{bmatrix} \quad (11)$$

Now for the least squares piece.

$$\mathbf{A}\mathbf{x}=\mathbf{b} \text{ and } \mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \text{ so } \mathbf{\Sigma}\mathbf{V}^T \mathbf{x} = \mathbf{U}^T \mathbf{b} \text{ or } \mathbf{x} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T \mathbf{b}$$

$\mathbf{\Sigma}^{-1}$ is the pseudoinverse. Its structure was detailed earlier.

Now applying equation (8) to the original right hand side yields,

$$\mathbf{x} = \begin{bmatrix} \frac{1}{9} & 0 & \frac{1}{9} \\ 0 & \frac{2}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{6} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{9} \\ \frac{1}{3} \\ \frac{7}{18} \end{bmatrix}$$

This is the least squares solution of minimal length.

Now the test, if \mathbf{x} is orthogonal to $N(\mathbf{A}^T)$, i.e., really the minimal length solution and \mathbf{u}_3 is an element of $N(\mathbf{A}^T)$ then orthogonality demands that their dot product equal zero.

$$\mathbf{x} \cdot \mathbf{u}_3 = \begin{bmatrix} \frac{2}{9} \\ \frac{1}{3} \\ \frac{7}{18} \end{bmatrix} \cdot \begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} \\ \frac{-2}{3} \end{bmatrix} = \frac{8}{54} + \frac{6}{54} - \frac{14}{54} = 0$$

This example is easily extended to matrices of greater dimension.

IV. COMPARISON OF METHODS

A. GENERAL

The procedures used for computational comparison were NAVSPACECOMs algorithms AMA03, AMA04, and AMA06 coupled with a process driver program and the standard SVD algorithms as taken from *Numerical Recipes* by Press, et. al., [7]. During research of the actual NAVSPACECOM code, it was found that *Numerical Recipes* was referenced repeatedly throughout. This observation set the precedence for applying the basic SVD codes of *Numerical Recipes* for computational analysis. This common ground should serve to standardize any resulting code development.

It should be noted that throughout chapter 15, *Numerical Recipes* [7] the SVD is recommended for least squares problems. Citing the following, "...solution of a least squares problem directly from the normal equations is rather susceptible to roundoff error." And, "In some applications, the normal equations are perfectly acceptable for linear least squares problems. However, in many cases the normal equations are very close to singular." The authors detail with significant correlation the precise difficulties that generally arise in Differential Correction. They go on further detailing the exact complication the routine AMA06 attempts to eliminate. The following excerpt from *Numerical Recipes* sums up the exact nature of this entire analysis:

A zero pivot element may be encountered during the solution of the normal equations, with Gauss-Jordan, in which you get no solution at all. Or, a very small pivot may occur in which case you typically get fitted parameters a_k with very large magnitudes that are delicately balanced to cancel out almost precisely when the fitted function is evaluated.

Why does this commonly occur? The reason is that, data do not clearly distinguish between two or more basis functions provided. If two such functions or two different combinations of functions happen to fit the data equally well - or badly - then the matrix A , is unable to distinguish between them and becomes singular. There is irony in the fact that least squares problems are both overdetermined (number of data points greater than the number of parameters) and underdetermined (ambiguous combinations of parameters exist). The ambiguities can be extremely hard to notice *a priori* in complicated problems.

The SVD gives exactly what we need. In the overdetermined system SVD produces a solution that is the best approximation in the least squares sense. In the case of the underdetermined system, SVD produces a solution whose values (the a_k 's) are the smallest in the least squares sense also what we want. When some combination of the basis functions is irrelevant to the fit, that combination is driven down to a small, innocuous value, rather than pushed up to delicately canceling infinities.

B. SENSITIVITY ANALYSIS

The following sections present information that details the inner workings of the individual numerical solvers. The concept to focus on deals with the aspect of matrix *sensitivity*. The following simplified example details the problem explicitly (see Nyhoff [16]).

$$\mathbf{Ax} = \mathbf{b} \text{ is } \begin{bmatrix} 2 & 6 \\ 2 & 6.0000003 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ 8.0000003 \end{bmatrix}$$

The solution is obviously $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Now perturb the right hand side very slightly.

$$\mathbf{Ax} = \mathbf{b}^* \text{ is } \begin{bmatrix} 2 & 6 \\ 2 & 6.0000003 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ 7.9999994 \end{bmatrix}$$

Now the not so obvious solution is $\mathbf{x} = \begin{bmatrix} 10 \\ -2 \end{bmatrix}$, where the very small perturbation in the right hand side has altered the solution on the order of 10^7 times the constant term of the second equation.

In performing rounding error analysis, we note that computers operate in floating point arithmetic. That is, they convert every problem into a “*nearby*” problem perform calculations and then convert back the answer. The symbol ε , called machine epsilon refers to the computer’s ability to distinguish between two consecutive binary representations of actual input values. When the relative representation of two consecutive numbers exceeds the computer’s ability to distinguish between them, ($\|a-b\| < \varepsilon$) we say an underflow has occurred in the calculation sequence.

The following method, known as *matrix perturbation theory*, helps to analyze the nature of error in order to minimize it through algorithm refinement. Errors are accumulated primarily from two sources. First there may be measurement error contained within the original input data, and second, the algorithm itself may cause error through its internal approximations while processing calculations. For excellent derivations of the most commonly required error measurement techniques, see Demmel [6].

The measure of the accumulated error is referred to as the *condition number* of a matrix. Simply put, the condition number indicates the precise level of sensitivity a matrix has relative to these types of very small perturbations.

To summarize, an example taken from Strang [3] is in order. It directly illustrates the short fall in the present NAVSPACECOM code methodology of scaling the input matrix by the value 25,000,000.

Begin by consider the linear system $\mathbf{Ax} = \mathbf{b}$, now perturb the right hand side by $\delta \mathbf{b}$. These errors might have come from the observational data or roundoff. The change is small but the direction of change can not be controlled. The solution is subsequently changed from \mathbf{x} to $\mathbf{x} + \delta \mathbf{x}$. Now our system has become $\mathbf{A}\delta \mathbf{x} = \delta \mathbf{b}$. We now must estimate the resulting perturbation $\delta \mathbf{x} = \mathbf{A}^{-1}\delta \mathbf{b}$. There will be a large change in the solution when \mathbf{A}^{-1} is large, i.e. \mathbf{A} is nearly singular. Now consider our symmetric matrix with positive eigenvalues where $0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$. Any vector $\delta \mathbf{b}$ is a combination of the corresponding unit eigenvectors $\mathbf{x}_1, \dots, \mathbf{x}_n$. Let ε indicate a very small change.

$$\text{If } \delta \mathbf{b} = \varepsilon \mathbf{x}_1 \quad \text{then} \quad \delta \mathbf{x} = \frac{\delta \mathbf{b}}{\lambda_1}.$$

The error in $\|\delta \mathbf{b}\|$ is amplified by $\frac{1}{\lambda_1}$, which is the largest eigenvalue of \mathbf{A}^{-1} . The

amplification is greatest when λ_1 is close to zero. Thus nearly singular matrices are the

most sensitive. The serious drawback with this measure of sensitivity appears when scaling is introduced. Multiplying the matrix by a large scalar also scales the eigenvalue by the corresponding amount. This makes the matrix appear much less singular. This rescaling can't however make an ill-conditioned matrix well. It is true $\delta \mathbf{x}$ will be smaller by the scale factor however so will the solution to $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. The relative error $\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|}$ stays the same. The factor $\|\mathbf{x}\|$ in the denominator normalizes the problem against such trivial rescaling. There is a corresponding normalization of $\|\delta \mathbf{b}\|$. The problem is to compare the relative change $\frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|}$ with the relative error $\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|}$.

At this point, I refer to the following theorem as taken from Strang, [3].

Theorem

For a positive definite matrix, the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ and the error $\delta \mathbf{x} = \mathbf{A}^{-1}\delta \mathbf{b}$ always satisfy,

$$\|\mathbf{x}\| \geq \frac{\|\mathbf{b}\|}{\lambda_n} \quad \text{and} \quad \|\delta \mathbf{x}\| \leq \frac{\|\delta \mathbf{b}\|}{\lambda_1}$$

Therefore the relative error is bounded by

$$\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\lambda_n}{\lambda_1} \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|}.$$

The ratio $c = \frac{\lambda_n}{\lambda_1} = \frac{\lambda_{\max}}{\lambda_{\min}}$ is called the condition number of \mathbf{A} .

This analysis can be applied to the example given previously.

C. ERROR ANALYSIS

The following analysis applies to the least squares system. The established error bounds hold regardless of solution method. A fundamental rounding error analysis follows directly from the previous sensitivity analysis. See Golub, [5] and Demmel, [6] for very complete presentations of both rounding and backward error analysis

The tradition definition of a vector or matrix norm holds throughout the following analysis. See Strang [3] as required. Where the symbol $\|\cdot\|$ appears in equation form, consistent use of the chosen norm is implied throughout. Condition number is as defined previously.

Given the linear system $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$. Where \mathbf{A} is $n \times n$ nonsingular, \mathbf{x} is the computed solution, and \mathbf{r} is the residual. Let \mathbf{x}_e be the exact solution and e the error.

$$\text{Then } \mathbf{r} = \mathbf{b} - \mathbf{Ax} = \mathbf{Ax}_e - \mathbf{Ax} = \mathbf{A}(\mathbf{x}_e - \mathbf{x})$$

$$\text{or} \quad r = Ae$$

Given A nonsingular then $e = A^{-1}r$. Taking consistent norms of the equations yields the following inequalities.

$$\|e\| \leq \|A^{-1}\| \|r\| \quad \text{and} \quad \|A\| \|e\| \geq \|r\|$$

Relating these inequalities bounds the error as follows,

$$\|A^{-1}\| \|r\| \geq \|e\| \geq \frac{\|r\|}{\|A\|} \quad (12)$$

Now the exact solutions form,

$$x_e = A^{-1}b \quad \text{and} \quad Ax_e = b$$

Where

$$\|x_e\| \leq \|A^{-1}\| \|b\| \quad \text{and} \quad \|A\| \|x_e\| \geq \|b\|$$

We can now bound the exact solution.

$$\|A^{-1}\| \|b\| \geq \|x_e\| \geq \frac{\|b\|}{\|A\|}$$

Now divide inequality (10) through by $\|x_e\|$ and form

$$\frac{\|A^{-1}\| \|r\|}{\|x_e\|} \geq \frac{\|e\|}{\|x_e\|} \geq \frac{\|r\|}{\|A\| \|x_e\|}$$

Substituting $\|x_e\|$ in the left denominator by $\frac{\|b\|}{\|A\|}$ and in the right denominator by

$\|A^{-1}\| \|b\|$ the expression for relative error now becomes,

$$\frac{\|A^{-1}\| \|r\|}{\frac{\|b\|}{\|A\|}} \geq \frac{\|e\|}{\|x_e\|} \geq \frac{\|r\|}{\|A\| \|A^{-1}\| \|b\|} \quad (13)$$

Where $\frac{\|e\|}{\|x_e\|}$ is the relative error.

Now recalling that the condition number of the matrix A can also be expressed as

$cond(A) = \|A^{-1}\| \|A\|$, which is required for analysis of methods that do not return eigenvalue information, equation (11) becomes,

$$cond(A) \left(\frac{\|r\|}{\|b\|} \right) \geq \frac{\|e\|}{\|x_e\|} \geq \frac{1}{cond(A)} \left(\frac{\|r\|}{\|b\|} \right)$$

This formulation is convenient in that it allows for testing of the maximum and minimum associated errors for any given output as initiated by the condition of the matrix system. For more specific information on relative error and on absolute error given

machine dependent performance information see Engeln-Mullges and Uhlig [7]. This analysis can be applied to the given test matrices.

D. COMPUTATIONAL EFFICIENCY

Flops, or floating point operations, are the measure of algorithm efficiency. While they do not necessarily indicate the actual processing time, as different computers do internal processing differently, they do give an excellent measure of the magnitude of total number of computer calculation required by an algorithm as related to the functional input.

In general, flop counts are obtained by summing the number of arithmetic operations from the most deeply nested algorithm statements. As an example of the accounting notice, a dot product operation of length n involves n multiplications and n additions. It therefore requires $2n$ flops. For matrix multiplications, the general form is

$$C(i, j) = A(i, k) * B(k, j) + C(i, j)$$

This process requires $2mnp$ flops where $C \in \mathbf{R}^{m \times n}$, $A \in \mathbf{R}^{m \times p}$, $B \in \mathbf{R}^{p \times n}$.

Applying the above flop counting procedure, the following flop count estimates are given for each algorithm as related to the matrix dimension [5].

ALGORITHM	FLOP COUNT
Gauss-Jordan	$mn^2 + \frac{n^3}{3}$
SVD (U, Σ , V)	$4m^2n + 8mn^2 + 9n^3$
SVD (Σ , V only)	$4mn^2 + 8n^3$ Minimum requirement for least squares

This does not take into account specialized storage techniques and factorizations when dealing with specific subprocess forms that may be optimized. In this respect, these estimates would be considered worst case. In some instances the total flop count of a particular subform may be cut by nearly half the indicated estimate. While these methods yield respectable approximations of the computational cost, they do not provide actual run time analysis. Specific algorithms can be made extremely efficient when optimized for a particular computer.

V. CONCLUSIONS

A. SUMMARY OF FINDINGS

Testing of the actual NAVSPACECOM code and the SVD code was accomplished through PC based Digital Visual Fortran 6.0. All NAVSPACECOM source code was compiled and linked using the visual Fortran development environment. As noted earlier SVD source codes were adopted directly from *Numerical Recipes* [6]. Driver programs for both software suites were developed as adaptations of the *LAPACK* [9] and *Numerical Recipes* source code for advanced linear algebra. Test matrices were generated using MATLAB and placed in standard text format for file upload by the driver programs. All Fortran source code was version 77. The computer system used was a Micron, 300 MHz, Pentium PII.

The test matrices indicated at Appendices A and B were used to determine how well the two routines compared when attempting to solve highly singular and known singular input matrices. As expected, the NAVSPACECOM code returns the $SING = 0$ error and does not continue further. The SVD algorithm on the other hand returns. Further, upon analysis of the factored structure, the reconstructed matrices are within one order of magnitude of the original input. The differences are due to roundoff error.

While the results presented in the output of the SVD algorithm may not look so appealing at first glance, it is important to note, that the present Gauss-Jordan,

NAVSPACECOM routines return no output at all. This is much more encouraging. As a minimum, even with the ridiculously singular nearly impossible input matrices, the SVD routine successfully returned the best approximation as the desired solution. At this point the differential correction subroutine now possesses an excellent starting point to continue processing this observational input set. The nonlinear method may then still converge. Had the NAVSPACECOM routine been allowed to process this set, the entire set would have been discarded based solely on the coincidental fact that the input matrix derivation is singular beyond the computer's ability to distinguish otherwise. The macro effect of this NAVSPACECOM code shortfall is that potentially accurately derived observational data is now discarded. This will certainly effect the long-term distribution of the accumulated error in the orbital element set.

B. RECOMMENDATIONS

The potential for improvement in NAVSPACECOMs differential correction process through integration of an SVD algorithm which replace existing subroutines AMA03, AMA04, and AMA06 exists. The only way to verify the magnitude of improvement, however, is to implement a test routine. The evidence presented herein indicates that further study should be pursued.

C. CONCLUSION

The results of this study are not all inclusive. Only the worst possible input matrix conditions were modeled. The frequency of occurrence of this type of input would

certainly have to be taken into consideration when deciding whether to upgrade the present process. Differential Correction involves highly complex series of numerical routines each with its own influence on the over all solution process. My findings indicate that some improvement may occur through reduced processing time and the numerical error of the actual values returned in the update state vectors. Additionally, it is worth pointing out that many modern statistical regression-fitting packages have begun to use more sophisticated linear algebra solvers for similar reasons as those addressed herein.

While at best the SVD algorithm is approximately 24 times more computationally expensive in flops, it has the advantage of not requiring cyclic preconditioning to start the solution process. The present NAVSPACECOM routine has the potential to cycle for significant periods prior to achieving $SING = 0$ stop criteria. If significant numbers of orbital element sets cause the AMA06 routine to process several times through before either failing or calling the solver, the relative difference in run time between the two methods may be very small. This aspect of the study will require actually run time information to determine the flop count delta.

SVD solutions exhibit less error in each iterate as a result of their "*best approximation*". It is possible that over time, the resulting thorough evaluation of the observational data by an SVD synthesizing algorithm could effectively decrease the error in actual known position information as measured by the difference in the orbital element set and the corresponding test data.

APPENDIX A

Test Case 1. Singular A Matrix

Highly singular A matrix: $\text{col}(4) = (\text{col}(1) + \text{col}(3))/2$ and $\text{col}(8) = (\text{col}(2) + \text{col}(7))/2$

Constructed random uniform (0,1)

Matrix A:

0.9688	0.1310	0.5620	0.7654	0.5979	0.0631	0.7666	0.4488
0.3557	0.9408	0.3193	0.3375	0.9492	0.2642	0.6661	0.8035
0.0490	0.7019	0.3749	0.2120	0.2888	0.9995	0.1309	0.4164
0.7553	0.8477	0.8678	0.8116	0.8888	0.2120	0.0954	0.4715
0.8948	0.2093	0.3722	0.6335	0.1016	0.4984	0.0149	0.1121
0.2861	0.4551	0.0737	0.1799	0.0653	0.2905	0.2882	0.3716
0.2512	0.0811	0.1998	0.2255	0.2343	0.6728	0.8167	0.4489
0.9327	0.8511	0.0495	0.4911	0.9331	0.9580	0.9855	0.9183

Symmetric, nxn matrix $P = A^T A$.

3.4538	2.2680	1.7824	2.6181	2.6412	1.9559	2.2782	2.2731
2.2680	3.0953	1.5425	1.9052	2.7916	2.2445	1.9391	2.5172
1.7824	1.5425	1.4978	1.6401	1.6543	1.0673	1.0142	1.2783

2.6181	1.9052	1.6401	2.1291	2.1478	1.5116	1.6462	1.7757
2.6412	2.7916	1.6543	2.1478	3.0720	1.8867	2.3445	2.5680
1.9559	2.2445	1.0673	1.5116	1.8867	2.8209	1.9602	2.1023
2.2782	1.9391	1.0142	1.6462	2.3445	1.9602	2.7791	2.3591
2.2731	2.5172	1.2783	1.7757	2.5680	2.1023	2.3591	2.4382

Decomposition Matrices:

Matrix U

-0.405972	0.466532	-0.366483	0.180020	-0.532409	-0.026839	-0.006785	0.408157
-0.387027	-0.208141	0.614596	-0.035119	-0.301571	-0.411911	0.408982	0.006625
-0.238695	0.389374	0.258549	0.159762	0.718607	-0.126578	-0.006750	0.408139
-0.322333	0.427960	-0.053982	0.169871	0.093150	-0.075934	0.013911	-0.816422
-0.404391	0.039450	0.240801	-0.441264	-0.010121	0.762960	-0.001406	0.000463
-0.326910	-0.511282	-0.089709	0.724015	0.092756	0.301448	-0.000586	0.000171
-0.346071	-0.284110	-0.593342	-0.385717	0.303011	-0.199784	0.408545	0.006797
-0.366547	-0.246130	0.010611	-0.210450	0.000580	-0.308105	-0.815805	-0.013969

Diagonal of Matrix Σ

16.942978	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
-----------	----------	----------	----------	----------	----------	----------	----------

Matrix V-Transpose

-0.405972	-0.387027	-0.238695	-0.322333	-0.404391	-0.326910	-0.346070	-0.366547
0.467637	-0.210013	0.388574	0.428113	0.038740	-0.511046	-0.282277	-0.246149
-0.370840	0.614798	0.254242	-0.058314	0.255496	-0.115354	-0.581027	0.016870
0.169917	-0.013430	0.165458	0.167672	-0.432943	0.720162	-0.406542	-0.210019
-0.575737	-0.269208	0.669202	0.201179	-0.065735	0.073310	0.312403	0.021403
0.261994	-0.244425	0.394398	-0.727229	0.369304	0.160414	-0.054058	-0.148875
-0.223596	-0.360038	-0.310865	0.334620	0.664486	0.262515	-0.175497	-0.265547
0.000238	-0.407559	0.000561	-0.000351	-0.001333	-0.000496	-0.407884	0.817022

Check product against original matrix:

Original P Matrix:

3.453800	2.268000	1.782400	2.618100	2.641200	1.955900	2.278200	2.273100
2.268000	3.095300	1.542500	1.905200	2.791600	2.244500	1.939100	2.517200
1.782400	1.542500	1.497800	1.640100	1.654300	1.067300	1.014200	1.278300
2.618100	1.905200	1.640100	2.129100	2.147800	1.511600	1.646200	1.775700
2.641200	2.791600	1.654300	2.147800	3.072000	1.886700	2.344500	2.568000
1.955900	2.244500	1.067300	1.511600	1.886700	2.820900	1.960200	2.102300
2.278200	1.939100	1.014200	1.646200	2.344500	1.960200	2.779100	2.359100
2.273100	2.517200	1.278300	1.775700	2.568000	2.102300	2.359100	2.438200

Product Matrix = $U \cdot \Sigma^*(V\text{-Transpose})$:

2.792422	2.662113	1.641832	2.217128	2.781547	2.248604	2.380400	2.521245
2.662113	2.537884	1.565216	2.113665	2.651745	2.143672	2.269318	2.403590
1.641832	1.565216	0.965332	1.303582	1.635438	1.322089	1.399580	1.482391
2.217128	2.113665	1.303582	1.760356	2.208493	1.785347	1.889991	2.001818
2.781547	2.651745	1.635438	2.208493	2.770714	2.239847	2.371130	2.511426
2.248604	2.143672	1.322089	1.785347	2.239847	1.810694	1.916823	2.030238
2.380401	2.269319	1.399580	1.889991	2.371130	1.916823	2.029173	2.149235
2.521245	2.403590	1.482391	2.001818	2.511426	2.030238	2.149235	2.276402

APPENDIX B

Test Case 2. Extreme Singularity

Singular P matrix: $\text{col}(1) = \text{col}(2) + \text{col}(3) + \dots + \text{col}(8)$

1.000000	0.900000	0.090000	0.009000	0.000900	0.000090	0.000009	0.000001
0.900000	0.900000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.090000	0.000000	0.090000	0.000000	0.000000	0.000000	0.000000	0.000000
0.009000	0.000000	0.000000	0.009000	0.000000	0.000000	0.000000	0.000000
0.000900	0.000000	0.000000	0.000000	0.000900	0.000000	0.000000	0.000000
0.000090	0.000000	0.000000	0.000000	0.000000	0.000090	0.000000	0.000000
0.000009	0.000000	0.000000	0.000000	0.000000	0.000000	0.000009	0.000000
0.000001	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000001

Decomposition Matrices:

Matrix U

-0.726830	-0.298545	0.377237	0.221507	-0.175290	0.150237	-0.127672	0.348738
-0.685806	0.302705	-0.443863	-0.223180	0.175290	-0.150237	0.127672	-0.348738
-0.037087	0.334128	0.811978	-0.190945	0.175288	-0.150237	0.127672	-0.348738
-0.003546	-0.836417	0.031208	-0.329137	0.174926	-0.150235	0.127672	-0.348738
-0.000353	-0.089017	-0.019792	0.864991	0.288233	-0.149928	0.127670	-0.348738

-0.000035	-0.008940	-0.002062	0.089682	-0.886388	-0.261431	0.127375	-0.348738
-0.000004	-0.000894	-0.000207	0.008994	-0.091411	0.899086	0.248332	-0.348622
0.000000	-0.000099	-0.000023	0.001000	-0.010180	0.102642	-0.916846	-0.385685

Diagonal of Matrix Σ

1.860190	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
----------	----------	----------	----------	----------	----------	----------	----------

Matrix V-Transpose

-0.676744	-0.731343	-0.084492	-0.003917	-0.000497	-0.000049	-0.000005	-0.000001
0.634569	-0.521247	-0.570616	-0.004328	-0.001723	-0.000168	-0.000017	-0.000002
0.187067	-0.224736	0.406687	0.865145	0.024989	0.002324	0.000231	0.000026
0.322910	-0.377922	0.708234	-0.499791	-0.038919	-0.001936	-0.000172	-0.000019
0.008663	-0.010371	0.016396	-0.041196	0.998458	0.030457	0.002268	0.000243
0.000000	0.000017	0.000174	0.001730	0.030609	-0.999175	-0.026540	-0.002122
0.000000	-0.000001	-0.000015	-0.000147	-0.001470	-0.026647	0.999247	0.028168
0.000000	0.000000	0.000001	0.000014	0.000138	0.001378	0.028215	-0.999601

Check product against original matrix:

Original Matrix P:

1.000000	0.900000	0.090000	0.009000	0.000900	0.000090	0.000009	0.000001
0.900000	0.900000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.090000	0.000000	0.090000	0.000000	0.000000	0.000000	0.000000	0.000000
0.009000	0.000000	0.000000	0.009000	0.000000	0.000000	0.000000	0.000000

0.000900	0.000000	0.000000	0.000000	0.000900	0.000000	0.000000	0.000000
0.000090	0.000000	0.000000	0.000000	0.000000	0.000090	0.000000	0.000000
0.000009	0.000000	0.000000	0.000000	0.000000	0.000000	0.000009	0.000000
0.000001	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000001

Product $U \cdot \Sigma^*(V\text{-Transpose})$:

0.914987	0.988808	0.114237	0.005295	0.000672	0.000067	0.000007	0.000001
0.863342	0.932996	0.107789	0.004997	0.000634	0.000063	0.000006	0.000001
0.046687	0.050454	0.005829	0.000270	0.000034	0.000003	0.000000	0.000000
0.004464	0.004824	0.000557	0.000026	0.000003	0.000000	0.000000	0.000000
0.000444	0.000480	0.000055	0.000003	0.000000	0.000000	0.000000	0.000000
0.000044	0.000048	0.000006	0.000000	0.000000	0.000000	0.000000	0.000000
0.000004	0.000005	0.000001	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000001	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

REFERENCES

- [1] Vallado, D., *Fundamentals of Astrodynamics & Applications*, McGraw Hill, 1997.
- [2] Danielson, D.A., and Canright, D., *Documentation of the Naval Space Command Differential Correction Process*, NPS Technical Report, in preparation.
- [3] Strang, G., *Linear Algebra and its Applications*, third edition, Harcourt Brace Jovanovich, 1976.
- [4] Goulub, G.H., and Van Loan, C.F., *Matrix Computations*, third edition, Johns Hopkins University Press, 1996.
- [5] Demmel, J. W., *Applied Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, 1997.
- [6] Press, W.H., et. al., *Numerical Recipes in Fortran*, second edition, Cambridge University Press, 1992.
- [7] Engeln-Mullges, G. and Uhlig, F., *Nnumerical Algorithms with Fortran*, Springer-Verlag, Berlin, 1996.
- [8] Nyhoff, L. R. and Leestma, S. C., *Fortran 90 for Engineers and Scientists*, PrenticeHall, New Jersey, 1997.
- [9] Anderson, E., Bai, Z., and Van Loan, C. F., *LAPACK User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center2
 8725 John J. Kingman Rd., STE 0944
 Ft. Belvior, Virginia 22060-6218

2. Dudley Knox Library2
 Naval Postgraduate School
 411 Dyer Rd.
 Monterey, California 93943-5000

3. Dr. Paul Schumacher1
 Naval Space Command
 5280 Forth Street
 Dahlgreen, Virginia 22448-5000

4. Lt. Col. David J. Vallado1
 HQ U.S. Space Command/AN
 250 S. Peterson BLVD, Ste. 116
 Peterson AFB, Colorado 80914-3180

5. Chairman Michael A. Morgan.....2
 Code MA/Mw
 Naval Postgraduate School
 Monterey, California 93943-5101

6. Professor Don Danielson5
 Code MA/Dd
 Naval Postgraduate School
 Monterey, California 93943-5101

7. Professor David Canright.....1
 Code MA/Ca
 Naval Postgraduate School
 Monterey, California 93943-5101

8. Capt. Patrick M. Marshall.....1
 4027 Prescott Dr.
 Martinez, Georgia 30907